# Sequence Labelling with CRF and RNNs

Nicolò Alessandro Girardini, Mat. number 203265
nicolo.girardini@studenti.unitn.it

***Abstract*—This is the report of the second project of the course Language Understanding Systems (LUS). It involves the comparison of different sequence labelling model, trained on a given dataset. The models implemented and confronted are CRF, LSTM and CRF on top of the LSTM. All the code and complete results can be found at the public repository [1]**

## I. Introduction

This project focuses on implementing models able to perform **IOB concept tagging** (Inside, Outside, Beginning) and confront their performance w.r.t. the previous project [2]. The dataset is **NL2SparQL4NLU** [3] which has as context movies: it is already divided into training and testing set. The resulting model will then be able to distinguish entities such as actors and movie names. The task has been achieved by implementing state-of-the-art models: **CRF (Confitional Random Fields)** a **RNN (Recurrent Neural Networks)** architecture, more specifically a **biLSTM** and finally the stacking of the two, so a CRF on top of the biLSTM.

The language used for this project is *Python*. The library to implement the CRF model is *sklearn-crfsuite* [4] while the NNs have been implemented using the *PyTorch* [5] framework. For the representation of words in the NN also two pre-trained embeddings have been used: *GloVe* [6] and *Spacy* [7]. For the CRF layer on top of the biLSTM **pytorch-crf** [8] has been used.

## II. Dataset Description

To help getting started with the task and then improving the models it is necessary to know some basic information about the dataset. It contains sentences within the context of movies and these sentences are in the form of a query. Each of the sentences' word is tagged with the **IOB** format:

- **O:** these are the words out of context
- **B-concept.type:** these are the words that indicate the beginning of a concept (actor, movie director, etc.). Concepts composed by only one word will have this as tag
- **I-concept.type:** these are the words inside the concept. The ending word of the concept will still have this tag

The data is divided in training and test set, where the test set is around one fourth of the dataset. This is important for the evaluation, as well with the fact that the vast majority of the tags are **O** tags, being around 72% both in training and testing sets. Another important fact to report is that these sentences, probably because they have a structure similar to queries, are much shorter than the average english sentence.

More information can be found at [3], while more in-depth analysis is found in the first project report [2].

## III. CRF Model

The CRF models are not generative models like WFSTs are, but they are **discriminative** models. This means that they do not model the joint probability of word-tag $P(X, Y)$ anymore, but they model directly the posterior probability $P(Y|X)$, making it a classification task, so to recognize the best fitting sequence.

They also combine two other approaches: **Max Entropy** and **Factorization**. The first allows to include in the model the observations, while keeping all the probabilities as uniform as possible. The factorization, instead, allows to express a probability distribution in terms of local factors: it is easy to understand while thinking of a graph, where the local factors are computed using the nodes connected directly to them.

Combining these two ideas gives the **CRF** model. The generalized version doesn't have a tractable solution, but the **Linear Chain CRF** has an exact solution and it is great to predict a sequence of words, while also conditioning on previous transitions (tag to tag). This also solves the problem of **label bias** of the MaxEnt model, where the transition probabilities are only dependent on the current state: CRFs instead consider all the possible transitions, modelling the whole probability of a label (tag) sequence given the observation. Formally:

$$p(x|y) = Z(x)^{-1} \prod_{t=1}^{T} exp(\sum_{k=1}^{K} \Theta_k f_k(y_t, y_{t-1} x_t))$$

$$Z(x) = \sum_{y} \prod_{k} \Theta_k f_k(y_t, y_{t-1} x_t)$$

where $Z(x)$ is the partition function used as a normalization, to ensure that $p(y)$ sums up to one (it is important to notice that it is global and input-dependent), while the exponential part takes in consideration both the observation probability and the transition from the previous label probability, since we are using the linear chain model.

The power of CRFs is also that $x$ doesn't have to only be a word, but it allows to use templates of features (i.e. use "past" words like in ngram modelling and "future" words).

### A. CRF Implementation

The most important thing in the implementation is the template of features used. The following have proven to be the ones giving the best results, where **-k** refers to the previous k words and **+k** to the following k words:

- **bias:** default bias to add, always 1
- **word:** the current word as it is
- **word[:2]:** prefix of size 2

- **word[-3:]:** suffix of size 3
- **word.isdigit():** whether the token is a number (boolean)
- **lemma:** the lemma the word corresponds to
- **postag:** postag of the word
- **postag[:2]:** prefix of the postag
- **-k:word:** previous words
- **-k:word.isdigit():** whether previous words are numbers
- **-k:postag:** postag of previous words
- **+k:word:** following words
- **+k:word.isdigit():** whether following words are numbers
- **+k:postag:** postag of following words

The first parameter is then the value of k, which is the **window** of tokens used. This will always be symmetrical.

The other parameters are the **hyper-parameters** of the CRF algorithm. These are evaluated with the **F1 measure**, using a randomized search in the space of the possible given values, which uses a 3-folds cross-validation. To evaluate the results during the search the **F1 score** has been used, only computed on the meaningful concepts (even if comprising *O* doesn't change the results significantly). These are the **algorithm** (either *lbfgs* or *l2sgd*) and its **regularization terms**, the **max iterations** limit and the **frequency cutoff**.

### B. CRF Results

The most important parameter in the implementation is the **window size** because it will change the features of the words. In Table I the results for different sizes are reported. The most important measure is the **F1 score**. More in depth-results and the exact hyper-parameters of the algorithm for these results can be consulted in the repository.

Table I
CRF RESULTS

| Window Size | Cut-Off | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|
| 1 | NO | 0.9432 | 0.8367 | 0.7938 | 0.8147 |
| 2 | NO | 0.9482 | 0.8538 | 0.8139 | 0.8334 |
| 3 | YES | 0.9434 | 0.8423 | 0.8075 | 0.8245 |
| 4 | NO | 0.9456 | 0.8494 | 0.8066 | 0.8275 |
| 5 | YES | 0.9414 | 0.8401 | 0.7993 | 0.8192 |

The best result is then obtained with **window size 2**, **without cutoff**, which was actually only useful with window size 3 and 5 (with minimum frequency 2). It can be observed that the results are not too different from one another and that the F1 score does not have a very fast decay when increasing the window. This could be due to the fact that the sentences are short and thus having a window covering almost the whole sentence could model the entire structure of it (even if this would probably be an overfit).

Keep in mind that this might slightly differ if you use the parameters in the results because of approximations.

## IV. LSTM MODEL

The second model is a **LSTM (Long Short-Term Memory)**, a specific type of **RNN cell**. The RNNs are very suited for the sequence labelling task, as they do not work like a feed-forward network, which takes the whole input and produces an output, but they pass the first element of the sequence to a cell which produces an output and it is itself connected to the following cell through a **state vector**, conditioning the subsequent output to the one just computed. This allows to take in account time and so sequences. The output of the net can then be used in two ways: the last output can be seen as a representation of the whole sequence, but, more importantly for this task, the outputs for each token form themselves a sequence.

Such RNNs have a **vanishing gradient problem**, meaning that when backpropagating the error though the gradient, it becomes too small and the weights do not update anymore. With this problem long-term dependencies are hard to capture. This is why gated units such as LSTMs and GRUs (Gated Recurrent Unit) were developed. They have similar performance and in this work the LSTM has been chosen. It is a cell that has three gates that decide which information to keep or to forget: the **input gate** which decides on the new information, the **forget gate** which decides on previous state of the cell and the **output gate** which decides on the output to pass to the other cells. A further improvement is the use of **biLSTM (bidirectional LSTM)** which makes use of another recurrent "layer" which starts from the end of the sequence, allowing the cell to take in account the future information as well.

### A. LSTM Architecture and Parameters

In this work a simple architecture has been used, which consists of the following layers:

- **Embedding layer:** layer needed to transform words into vector representations. The input dimension is equal to the vocabulary length and the output dimension is in all cases 300
- **LSTM layer:** a single recurrent layer. Bidirectional has been tried as well. The hidden dimension is 150 and half of it when bidirectional
- **Linear layer:** used to map the information learned by the LSTM layer to the concepts space, which has dimension 41
- **Log-Softmax layer:** used to compute the output log-probabilities of each of the concepts

To compute the error the **Negative Log-Likelihood Loss** has been used, which is good when dealing with multiple output labels (multiclass classification task). The optimizer is the **SGD (Stochastic Gradient Descent)**.

Before feeding it to the network, the data has been split into **batches** in a specific way. Batches have a maximum dimension of **50** and in each batch only sentences with the same length are present. This allows to avoid handling padding and performing useless computations on it, while keeping the benefits of mini-batch training. Bigger and smaller batches have been tested, but this was the best size for the performance-accuracy trade-off.

The most important preparation step, though, is the choice of the **word embeddings**. Three different possibilities where explored: **default embedding, GloVe embeddings** and **Spacy embeddings**. The step firstly needs to generate a vocabulary mapping words to indexes, which have to be consistent with the indexes of their corresponding vectors when the pretrained

embeddings are used. In the case of the default embedding, a vocabulary is generated only from the training set words. The embedding layer uses the weights of GloVe and Spacy models without updating them (they have already been trained), while with the default embedding it updates and learns the best embedding from the training data: the embedding dimension is **300**, to keep it equal to the pretrained models.

An important parameter is the **number of epochs**. This had to be tuned to balance good result and overfitting. Figure 1 shows the error, the F1 score of the training set and test set though epochs (until 150) for the best configuration. After 100 epochs there is no clear F1 improvement for the test set and almost no improvement even for the training set, while the loss has not very significantly improved. Even if overfitting does not seem an issue by looking at the test results, this might be due only to the very specific field this dataset has as context (movies). The chosen value was then **100** and the results where better for certain embeddings and worse for others, demostrating that there is no real correlation between increasing epochs and better results.

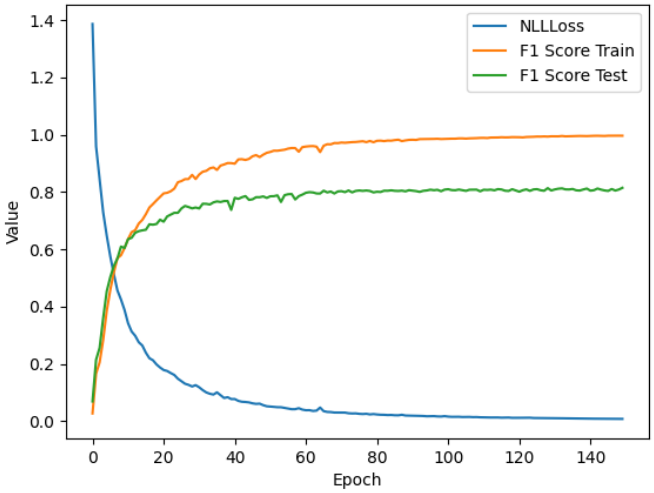


Figure 1. Learning Curve for the best parameters along 150 Epochs

As already said, **hidden representation** has dimension **150**. It is the exact half of the embedding dimension and as we see in the figure, it allows the training curve to be pretty smooth. Having a lower value lowers the F1 score, while increasing it too much simply makes the training more unstable, with significant drops in F1 score, thus the decision on 150.

The last discussion is whether to use lemmas or not. It is not needed with the pretraining embeddings as they already account for that in the vectors and actually the scores are a bit lower. Instead for the default encoding it significantly improved the results so lemmas will be used only for that.

Some parameters will not be discussed in depth: optimizer parameters only made the model reach the solution faster and had no significant impact on the overall performance, so it looks like the network is reaching a global optimum.

### *B. LSTM Results*

The main measure used to evaluate the model is always the F1 score and the main comparisons possible are between the word embeddings used and whether or not using a biLSTM improves the results. All the combinations are reported in Table II

Table II
LSTM RESULTS

| Embedding | Bidirectional | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|
| Default | NO | 0.9134 | 0.6557 | 0.6929 | 0.6738 |
| GloVe | NO | 0.9306 | 0.7115 | 0.7571 | 0.7336 |
| Spacy | NO | 0.9307 | 0.7153 | 0.7599 | 0.7369 |
| Default | YES | 0.9327 | 0.7893 | 0.7553 | 0.7719 |
| GloVe | YES | 0.9482 | 0.7995 | 0.8148 | 0.8071 |
| Spacy | YES | 0.9512 | 0.7970 | 0.8203 | 0.8085 |

The first thing to notice is that the **biLSTM** is always outperforming the LSTM by a huge amount. This means that knowing the "future" of the sequence really strengthens the ability to learn the sentences structure (which was also present in the CRF that has even better results).

For what concerns the embeddings, the best one in this setting is **Spacy**, with **GloVe** having very similar performances. The **Default** is always the worse one: this might be due to two factors. The first is that the pretrained embeddings have been trained on a much broader vocabulary with many different contexts for a long time, while the default one is just trained in this model and only has a movie context: the pretrained are then able to represent words much better. The second is that the default is only trained on the training set, as the test set cannot be used before the training is complete (or during the training to check how much the model improves): this really limits the words he can represent and it has to represent all unknown words (which are much more than the pretrained) with the same vector.

Overall the results were pretty close one another (with the exception of the results for the default embedding) even when using different optimization parameters and layer sizes, so using a biLSTM gives pretty consistent results. Still, the best result with the chosen parameters was achieved by using the Spacy embeddings.

## V. LSTM AND CRF MODEL

It is clear in the literature (e.g. see [9]) that stacking the two previous models used in this work is a simple way to improve the results of the task. This is possible because it allows to learn a good representation of the state with the LSTM and then use the power of the CRF to predict the concept. It has been tested using the simple LSTM developed to see if even with its simplicity it outperforms the CRF model. The stacking is possible thanks to pytorch-crf, which is module for handling the CRF as if it was a layer of the network.

The model in this work simply adds the **CRF layer** on top of the best network configuration found in the previous model. The softmax layer at this point is useless and we put the CRF

layer on top of the linear layer (all the already existing layers keep their dimensions).

The training of the network is a bit different: while before the output was a softmax layer, this time the output of the forward pass is directly the **mean Negative Log-Likelihood** computed by the CRF layer which is used as a loss. To be able to compute the F1 score it is performed a prediction pass as well, which, thanks to the CRF layer, can also return the optimal tags computed. This is the method used afterwards to predict the test concepts.

The table Table III reports the results for each of the different embeddings.

Table III
LSTM+CRF RESULTS

| Embedding | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Default | 0.9335 | 0.8473 | 0.7580 | 0.8002 |
| GloVe | 0.9552 | 0.8565 | 0.8478 | 0.8521 |
| Spacy | 0.9555 | 0.8533 | 0.8478 | 0.8506 |

The CRF layer was able to improve significantly the overall performance of the model, living up to the expectations: it is worth noting that during the training the F1 score on the training set reaches **0.9998**, even if it starts from a much higher loss than the previous model (because of the CRF). Once again the **Default** embedding does not perform as well as the pretrained models, with the best one this time being **GloVe**, demonstrating how there is not much difference between the two.

## VI. POSSIBLE IMPROVEMENTS

It is clear that there is room for improvement. For what concerns the CRF more features could be engineered and they could be more sophisticated, e.g. use word vectors to compute word similarity with the words in the window and others.

For the networks this work makes use of simple architectures with relatively small hidden representations and few layers. More complex architectures can significantly improve the result as well, for example with the introduction of convolutions and dropout layers. More data can also help in the training and allow to use bigger batches and more training epochs.

## VII. CONCLUSION

The models have been built correctly. As expected from NNs models they where able to perform pretty well, even with a simple architecture. It is still important to compare all the best results in this work, along with the results of the WFSTs of the previous project.

Table IV
MODELS RESULTS COMPARISON

| Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Base WFST | 0.9269 | 0.7851 | 0.7434 | 0.7637 |
| Advanced WFST | 0.9399 | 0.7893 | 0.8103 | 0.7996 |
| CRF | 0.9482 | 0.8538 | 0.8139 | 0.8334 |
| biLSTM | 0.9512 | 0.7970 | 0.8203 | 0.8085 |
| biLSTM+CRF | 0.9552 | 0.8565 | 0.8478 | 0.8521 |

It can be said that models based on **WFSTs** are outdated and do not perform as well as the state-of-the-art models, even if the latter have to be a bit refined before having a significant difference in performance with the first (and they also usually take more time to train). The **biLSTM** shows this: being a very simple model its performance is only slightly better than the more refined WFST model, but they still demonstrate a big representational power even with the basic architectures. The **CRF** confirms to be one of the few algorithms that can stand his ground against NNs by outperforming the simple biLMST (it could also probably even be improved). The stacking of these two latter models, **biLSTM+CRF**, outperforms every one of them and can be considered a state-of-the art model, thanks to the CRF layer being able to make use of the information learned by the biLSTM layer.

## REFERENCES


[1] "Github repository of the project," https://github.com/Svidon/LUS_Project2, Accessed: 2020-08-26.
[2] "Github repository of the previous project," https://github.com/Svidon/LUS_Project1, Accessed: 2020-08-26.
[3] "Nl2sparql4nlu dataset," https://github.com/esrel/NL2SparQL4NLU, Accessed: 2020-08-26.
[4] "Sklearn crf suite," https://sklearn-crfsuite.readthedocs.io/en/latest/, Accessed: 2020-08-26.
[5] "Pytorch," https://pytorch.org/, Accessed: 2020-08-26.
[6] "Glove embeddings," https://nlp.stanford.edu/projects/glove/, Accessed: 2020-08-26.
[7] "Spacy embeddings," https://spacy.io/models, Accessed: 2020-08-26.
[8] "Pytorch-crf," https://pytorch-crf.readthedocs.io/en/stable/, Accessed: 2020-08-26.
[9] J. Gobbi, E. Stepanov, and G. Riccardi, "Concept tagging for natural language understanding: Two decadelong algorithm development," 07 2018.